# Modelling Industry 4.0 based Fog Computing environments for Application analysis and deployment

Nandor Verba[a,*], Kuo-Ming Chao[a], Jacek Lewandowski[a], Nazaraf Shah[a], Anne James[b], Feng Tian[c]

[a] *Faculty of Engineering, Environment and Computing, Coventry University, Coventry, United Kingdom*
[b] *Department of Computing and Technology, Nottingham Trent University, Nottingham,United Kingdom*
[c] *Systems Engineering Institute,Xi'an Jiaotong University,Xi'an, China*

**Abstract**

The extension of the Cloud to the Edge of the network through Fog Computing can have a significant impact on the reliability and latencies of deployed applications. Recent papers have suggested a shift from VM and Container based deployments to a shared environment among applications to better utilize resources. Unfortunately, the existing deployment and optimization methods pay little attention to developing and identifying complete models to such systems which may cause large inaccuracies between simulated and physical run-time parameters. Existing models do not account for application interdependence or the locality of application resources which causes extra communication and processing delays. This paper addresses these issues by carrying out experiments in both cloud and edge systems with various scales and applications. It analyses the outcomes to derive a new reference model with data driven parameter formulations and representations to help understand the effect of migration on these systems. As a result, we can have a more complete characterization of the fog environment. This, together with tailored optimization methods than can handle the heterogeneity and scale of the fog can improve the overall system run-time parameters and improve constraint satisfaction. An Industry 4.0 based case study with different scenarios was used to analyze and validate the effectiveness of the proposed model. Tests were deployed on physical and virtual environments with different scales. The advantages of the model based optimization methods were validated in real physical environments. Based on these tests, we have found that our model is 90% accurate on load and delay predictions for application deployments in both cloud and edge.

*Keywords:* Cloud Computing, Fog Computing, Application Model, Migration

## 1. Introduction

The concept of Industry 4.0 provides new means of state of the art IT and manufacturing technologies integration through cybernetics, in order to advance automation of the

*Corresponding author
*Email address:* verban@coventry.ac.uk (Nandor Verba)

manufacturing systems and help to improve product quality, production efficiency, condition monitoring and decision making [1, 2]. Within this concept, machines become connected with humans through computer systems to work in a coordinated way to automate data acquisition, sharing and exchange among the physical and virtual worlds.

The wide spread availability and affordability of sensors and wireless networks and the accessibility of high speed Internet make real-time multiple parameters monitoring and control of manufacturing process possible in a way that was not possible before [3]. This leads to a great number of sensors being deployed to physical machines which in turn generates a large volume of data that requires computationally intensive analysis and interpretation for decision-making purposes. The resulting decisions, whether made by human or software, often require to be transformed to control signals for actuators to operate the machine in the physical world. This then creates a loopback to the sensor system as new sets of data are collected and sent back for further analysis, reflecting changing machine states over time.

This type of system based on Cyber-Physical System (CPS) is a facilitator for realising the concepts of Industry 4.0. It enables computational algorithms and physical components to interact with each other through real-time monitoring and control to improve productivity [4, 5]. Yet, as stated in [6] traditional servers with limited capacities may not be able to cope with the new challenges in terms of scalability and complexity of such systems. In turn, the cloud with Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) provides a promising integrated solution [7, 8] to address these challenges.

However, the cost of using cloud services and the latency between the edges of the network and the cloud could hinder its application to time critical applications. Furthermore, due to health and safety issues, some machines cannot be operated remotely and can only be operated within particular boundaries at the premises [9]. Besides, in some cases the control software and the machine are bundled together due to security, license and driver requirements, so it cannot be run on the cloud or other machines. On the other hand, the computational power or other resources in the controllers may have spare capacity to process additional tasks. The study of assigning these existing extra resources to time sensitive tasks and overloaded computer systems becomes an interesting alternative to cloud processing [10, 11]. Such systems as part of a network infrastructure could efficiently deal with front line demands [12].

The fog environment is an extension of the traditional cloud to the edge of the network including both cloud and edge resources. In Cloud computing environments, we consider the physical machines to have mostly homogeneous builds with network latencies inside clusters small enough that the deployed location of applications is not considered. In Fog environments, we consider locality to be an important issue, due to the heterogeneous build of nodes, the higher communication latencies inside a wireless network and the physical lock-in of device resources. These differences between Fog and Cloud approaches introduce the need for a new workload descriptor for a PaaS based Fog. Furthermore, in CPS systems we need to consider the importance of device to application communication. These require the formulation of a new Application and Gateway Model as well as new optimization approaches that can address the differences between these systems.

So far, only limited numbers of optimisation and load balancing approaches such as [13]

have been proposed in the context of fog computing and CPS. The existing approaches [14], however, focus on the design of optimisation methods using different simulated parameters such as workload, power consumption and virtual machine distribution. These approaches also lack physical environment testing, only consider theoretical formulas for delay and processing that can only be implemented with full knowledge of the applications and the systems that is not always possible. There are few studies [15] on application properties such as coupling nature between devices/drivers and controller/gateway. Such properties can lead to extra communication latency, depending on distance of device to gateway. The existing approaches [14] are mainly interested in overall or average workload balancing without considering individual message response time, which can be crucial for CPS.

The main contribution of this proposed paper is a new reference model with data driver profiling and parameter formulation. This model allows us to evaluate the load and delay of applications on our systems and also estimate the effect of changes and migration on these systems. We consider the Load and Delay components as Reliability, Energy use and Costs can be derived from the system load and messaging components. The model is supported by the workload profiles obtained from the experiments to increase its applicability. To accommodate this model, we propose the the Reliability Optimization (RO), Delay Optimization (DO) and Constrained Delay Optimization (ConstDO) functions and compare their performance. We evaluate these methods on our platform focusing on how well they improve overall delays, reliability and reduce constraint violation through both heuristic and deterministic optimization methods. We assess our models through the precision of their predictions in both single, bundled and migrated deployments. The objective functions are assessed based on how well they improve the app delays, gateway reliability and reduce the constraint violations for each gateway.

### *1.1. Scenario*

A fog computer in this paper is a gateway, part of a local cluster network, which is responsible for managing a job shop including a set of machines, sensors and software systems. The gateways, as a self-contained low cost computer system have limited computational resources and consume less energy than the cloud or servers, so they are deployed to the sites close to the endpoints/edges in a large scale distributed control system or CPS [16, 17]. The data generated by the machines and sensors, can be aggregated, compiled and analysed in the local fog computing cluster, rather than in the cloud, thus reducing bandwidth and decreasing the response/delay time. The fog computing architecture could provide an effective and efficient solution for large data-generation and computational-resource-requiring time-sensitive IoT/CPS applications [10, 18, 19, 20]. In such a fog environment, the non-time sensitive requests, which do not require real-time responses, can still be processed in the cloud, while the edge can respond to the urgent events triggered by machines or sensors that it monitors and controls. Similar architectures have been previously adopted to meet the time-constraint requirements of different applications [18, 21].

Consider the example in Fig. 1 based on use-case scenarios presented in [22]. Here, gateways monitor temperature and vibration in several CNC drilling machines with sensors in a job shop, and operate them accordingly. The data, which requires substantial computational

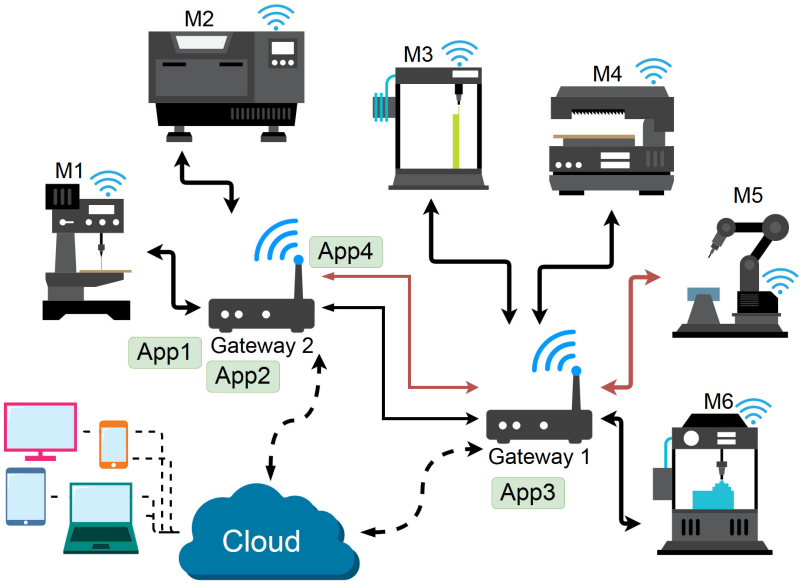



Figure 1: Deployment Scenario

resources to process, is sent to the cloud for analysis of machine conditions for maintenance and production performance. The data analytic tool may identify some time-sensitive conditions in some machines that need to be monitored closely and delegates the corresponding gateway to carry out the task. The gateway checks the indicators in real-time to examine if the current temperature and vibration rates collected from the sensors are greater than the predefined thresholds. If yes, the gateway triggers an instant warning and the machine stops.

In the initial scenario, the gateways and cloud can meet these requirements. But then a new set of machines (M5 and M6) are introduced to the job shop to increase production capacities. Once new machines are in operation, the allocated gateway (Gateway 1) cannot cope with these extra tasks and its processing delay time increases to a value that violates the applications constraint. To maintain efficiency, an optimisation method is applied, which decides to migrate application A4 to Gateway2 so that the real-time constraints are met, and that the delays of the system are reduced.

## 2. Background

Several CPS clouds have been proposed to enhance the functionalities and alleviate some barriers in CPS development [8]. However, latency is one of the inherent challenges in cloud computing, and this issue limits its use in time-sensitive applications. CPSs often work in a dynamic environment having a mixture of urgent, unexpected, and periodic events with hard and soft real-time constraints. This means the cloud approach may be inadequate for CPSs [23].

The core concept behind Fog Computing optimization is to improve efficiency of resource utilization thrughout the system. Load and Delay optimization as suggested in [24] is

imperative for time-sensitive CPS applications and has become an important field of research for managing resources in Fog environments. The use of OSGI and similar resource sharing platforms for application deployment compared to Container and VM based solutions offers new possibilities but also creates challenges due to increased interdependence.

### *2.1. Related Work*

Various resource management strategies and algorithms have been studied for decades in a variety of scenarios and it is well understood area in general distributed systems and cloud computing. However recent years have witnessed that researchers are moving their focus towards load balancing optimisation for cloud-fog systems. The key question in load balancing optimisation is how to allocate jobs on various machines so that each receives its fair share of resources to make progress while providing good performance [25].

The studies conducted by [26] and [27] provided comprehensive survey of various techniques for load balancing optimisation for cloud computing. The load balancing optimisation mechanisms used in managing resources in the cloud computing can be broadly divided into virtual machine (VM) based, QoS, and energy etc.. Hu et. al. [28] presented an algorithm for load balancing optimisation of VM resources by using a genetic algorithm. Their proposed algorithm attempts to reduce migration cost of VMs using historical data and current state of the system. Zhao et.al [29] presents the design and implementation of an algorithm that employs the Pareto dominance theory and simulated annealing to achieve a long-term efficient power saving and load balancing optimization. Dhinesh et. al. [24] proposed an algorithm to achieve load balancing across VMs in order to maximise and balance the priorities of tasks so that the amount of waiting time of the tasks is minimal [4]. A task scheduling algorithm for load balancing optimisation based on QoS, proposed in [30], computes the priority of tasks based on some specific attributes and evaluates the completion time of each task and then schedules each task onto a resource, which can complete the task according to the task priority. In [31] the authors propose a new load balancing optimisation method that uses Particle Swarm Optimization to balance system load by transferring tasks from an over-loaded VM to less loaded one.

Through application of various algorithms, these approaches revolve around optimal task distribution on various VMs or VM migration to achieve effective load balancing optimisation. The underlying characteristic of these algorithms requires their customisation to be applicable in fog computing settings.

The fog computing model extends the cloud load balancing problem from cloud resources to include edge device resources. The cloud load balancing optimisation methods cannot be applied in fog computing due to the fundamental difference between these computing models. The existing cloud optimisation methods for load balancing have shortcomings in terms of system hierarchy and load forecasting, which cannot be applied to the dynamic and P2P architecture of fog computing [32]. The fog computing dynamic load balancing optimisation mechanism provided in [32], through graph partitioning and clustering, assigns tasks to VMs according to the resource requirements of the tasks. Our proposed research, on the other hand, focuses on application migration.

In [10] a framework has been proposed to investigate the trade-off between power consumption and delay in the cloud-fog environment. In contrast our proposed method focuses on fog subsystem load balancing, considering task assignment on various gateways and service migration.

Furthermore, Verma et. al. [33] proposed a load balancing optimisation algorithm, which uses a data replication technique for maintaining data in fog networks with an attempt to reduce overall dependency on big data centers. The approach does not consider quick response to emergency messages, which is at the heart of our proposed approach.

Application latency in fog computing has been addressed by VM migration [34]. Their proposed approach focuses on allocation of VM to each mobile user, and migrates these VMs based on an objective function to optimise the result for both service providers and service consumers. Unlike ours, their approach assumes the availability of sufficient resources for hosting virtual machines and does not consider device limitations and service level migration.

In [35] an architecture is proposed to integrate fog computing and SDN (Software Defined Network) to IoV (Internet of Vehicle) to improve the latency of sensitive services. This approach is highly domain dependent and uses particle swarm optimisation to decrease service latency.

Table 1: Features of Fog Models and Algorithms

| Article | Task Oriented | Migration | VM Migration | Driver Coupling | Message Routing |
|---|---|---|---|---|---|
| Ningning et al. [32] | X | – | – | – | – |
| Bittencout et al. [34] | – | – | X | – | – |
| Deng et al. [10] | X | – | – | – | – |
| Verma et al. [33] | X | – | – | – | – |
| He et al. [35] | X | – | – | – | – |
| Our method | X | X | – | X | X |

The approaches for load balancing optimisation in fog computing, discussed here, do not consider the characteristics of the application and work on a common assumption, that all control and analysis applications are placed on all fog nodes. This assumption cannot be true for fog nodes with limited resource capacity.

Our proposed optimisation algorithm builds on extant heuristic load balancing optimisation approaches and performs load balancing optimisation through application migration, tasks assignment, emergency messages, heterogeneity of nodes and tight coupling of software drivers and sensors. The gateways/edge devices used in our application have limited resources and they have tendency to get overloaded quickly compared to cloud resources, which makes it imperative to have proactive optimisation mechanisms for load balancing to safeguard time-critical applications. Table 1 provides the comparison of the features of the extant fog computing algorithms for load balancing optimisation and our proposed algorithm. The parameter task orientation means that the load balancing optimisation algorithm operates on tasks to balance the load on various nodes. Application migration

and VM migration indicates that load balancing optimisation algorithm operates on service migration or VM migration to optimise load balancing.

Some sensor devices are tightly coupled with software drivers installed in gateways and the message still gets routed through the associated driver in the event of service migration. This in fact results in increased message paths. Driver coupling is not relevant to VM migration, where the whole VM gets migrated. Message routing through the original path is relevant to the application, where there is tight driver coupling and messages must be routed through specific drivers no matter where the application is deployed.

## 3. Platform Description

In this section, we describe our Fog of Things platform that is compliant with the Fog Computing paradigms and requirements presented in [36]. This platform supports the paradigms of Fog of Things by virtualizing devices and treating them as resources while providing a platform for application deployment and migration through message translation and routing.

### *3.1. Fog of Things Gateway Platform*

The Apache Karaf based Fog of Things Gateway that we proposed in [36] represents a PaaS approach to device orchestration and resource use at the edge of the network. Fog Computing paradigm considers processing, networking and storage as resources as in Cloud computing but extends these to the Edge of the network. This allows for the movement of tasks from the cloud down to the edge of the network and back based on requirements. Our platform adds to this paradigm by considering connected devices as resources of the gateway that are made available to applications through a messaging level virtualization. The device, storage, cloud, region and other resources request are translated to allow for protocol agnostic communication with the applications as well as migration. This virtualization layer allows the apps to have a uniform view of all available resources regardless of their location and protocol. The same virtualization can be found in the application layer, where messages to applications that reside on other gateways are translated and brokered through the messaging system giving the applications the illusion of one big virtual environment.

Modular application deployment, protocol agnostic messaging together with multi-cloud tenancy, or multiple clouds having access to one gateway, allows the gateway to better represent the horizontal integration requirements of Industry 4.0 [6] by enabling applications from multiple providers to interact, forming composite services, as well as allowing data to be sent and received from multiple cloud providers that may offer a varying set of services for data processing, device monitoring or control.

### *3.2. Migration and Message Routing on the Platform*

The message routing on the gateway is the backbone of the protocol agnostic messaging for drivers as well as the mechanism we use for the migration of applications between gateways. Migrated applications need to maintain their full functionality, being able to access

data in storage and all connected devices while being able to communicate with peers in the region, cloud and other services on the gateway.

The migration process requires all messages going from and to the application to be routed to the new host gateway. This gives the app the illusion of still being on the same gateway, without needing to reconfigure or rewrite its code.

The messaging service uses federated connections to forward messages between peers. The connections between gateways by default are in a star topology that allows each gateway to access each other directly, reducing latencies and hops but increasing overhead on larger systems. Other topologies can be designed per application environments.

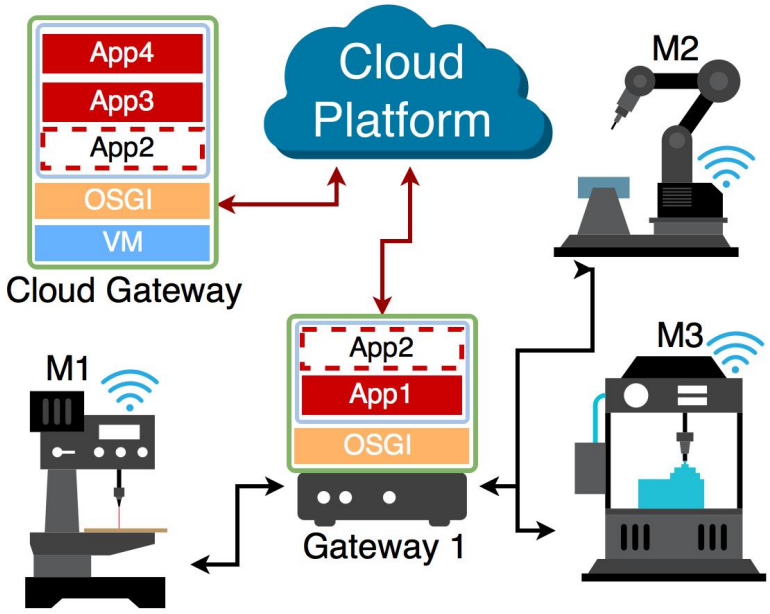



Figure 2: Migration

An example of how this routing is done can be seen in Fig. 2 where Application 2 is moved from the cloud VM to the gateway, this changes the run-time characteristics of the Application, the new environment having different latencies, different load and processing capabilities.

An application is migrated by deleting it or stopping it on the host gateway, re-configuring the existing routes of the application to be sent to the new host, adding the routes on the new host to the application container and then finally deploying and starting the application in the new hosts container. This is done through the configuration file of the application.

### *3.3. Application and Gateway Monitoring*

The monitoring on the gateway is done by two components. The first is inside the application container and monitors all application messages, even inter-app messages as well as the total CPU usage of the applications threads. This component sends a message to the gateway monitor which looks at a wider range of parameters but takes a more general

look at messages. The second component creates a summary file which is saved to the database periodically.

The monitor in the application container is able to retrieve information on messages sent to and from each application to any drivers, cloud connections, regions and resources. Furthermore, it reads information on the CPU usage of every application. The gateway monitor has a more general view of the messaging as it shows all messages routed from all components without information on individual users/applications. This monitor can also give information on individual application storage use, gateway load, RAM use and CPU usage on the system. The load is adjusted to the processor count of the platform.

### *3.4. Cloud Control, Migration and Optimization*

The gateways are configured to work with a centralized control system, where they receive management commands through configuration files, which contain all aspects of migration, deployment and clustering.

For the deployment optimization, we use a centralized control policy that allows the cloud to analyse the monitoring data from the gateways and deploy changes on required systems. This is done to decrease their impact on the platform itself.

The triggering policy for the optimization in the testing environment is a manual one, while the system would support scheduled triggering as well as parameter based triggering, where we look at certain key parameters and trigger the optimization if they cross certain values. Another method is gateway event based, where the gateway notifies the cloud of any events that modify the runtime parameters such as added or modified device, region and cloud connections.

## 4. Application and Gateway Models

The application model attempts to estimate the functioning parameters of an application based on the limited information we have on them to make estimation and optimization possible and estimate its effects. The model attempts to calculate the load of the application and its connected devices on the gateway which is used to estimate the effects of migration. To be able to measure the total delay of the device messages we use test drivers that allow us to send messages to the application container at a constant rate and measure the actual return time of messages. The processing delay is measured by the application itself. We do not consider the delay between drivers and physical devices, because these delays cannot be improved by the system and are subject to the adopted protocols and underlying connections.

$$T^P = I^P IPS^{Gw} \tag{1}$$

The equations are based on (1) which is the standard formula for processing time calculation, where $T^P$ is the execution time, $I^P$ is the Instruction Count while the $IPS^{Gw}$ is the gateway specific Instructions Per Second capability. Our equations which we define in the next subsections aim to adjust this formula to account for multiple processes running and the load of the gateway. The Instructions count $I^P$ is analogous to the Unit Load $L^u_{ij}$ presented in (7) while the $IPS^{Gw}$component is analogous to the $k1$ and $k2$ constants from

(9). The differences in Instruction per second capability of the systems are adjusted using the processing capacity $P_j^{Cap}$ and processing speedup $P_j^{Speed}$ which is based on Amdahl's law [37] where we consider that most of the code designed for IoT systems is sequential so a multi-core system will increase the processing capacity of our system but only a more performant processor will speed it up.

### *4.1. Gateway Load*

The gateway load is measured by the monitoring component on the gateway and is measured as the total CPU usage in the system in (%). The gateway has two types of overhead, the first type is generated by maintaining the cluster connections and background applications. The second is generated by device message processing by their respective drivers. Both are constant to a gateway and are not improved by migration.

We consider that the Gateway Cluster consisting of Gateways $G_j$ having $j$ denoting the gateway number containing Applications $A_{ij}$ defined as application number $i$ owned by gateway $j$.

The Gateway processing speedup $P_j^{Speed}$ is calculated in (2) by comparing the execution time $T_j^{Gw}$ of a bit of code on the gateway to the reference value $T_{Ref}^{Gw}$ which was run on our node with a $P_{Ref}^{Speed}$ of 1.

$$P_j^{Speed} = \frac{T_j^{Gw}}{T_{Refj}^{Gw} \; P_{Ref}^{Speed}} \tag{2}$$

The Gateway processing capacity $P_j^{Cap}$ is calculated in (3) by comparing the measured application load of a known reference application and comparing it to the value measured on the reference gateway that has a processing capacity $P_{Ref}^{Cap}$ of 1.

$$P_j^{Cap} = \frac{L_{ij}^{a}}{L_{iRef}^{a} \; P_{Ref}^{Cap}} \tag{3}$$

The Gateway Load $L_j^{Gw}$ the sum of Application Loads $L_{ij}^{A}$ and the Base Load $L_j^{B}$ the gateway in (4) and is measured in % of the CPU usage. This variable is directly measured through the monitoring component.

$$L_j^{Gw} = L_j^{B} + \sum L_{ij}^{A} \tag{4}$$

As defined in (5), the Base Load $L_j^{B}$ is the processing power used by system/background processes and drivers which are considered constant throughout migration process.

$$L_j^{B} = L_j^{Ide} + \sum L_k^{D} \; \lambda_{jk}^{Gw} * P_j^{Cap} \tag{5}$$

The Idle Load $L_j^{Idle}$ in (5) is the % CPU of the gateway $j$ at rest without any message passing or an application related activities. We consider this to be constant on every gateway, regardless of applications or connected devices or peers.

The Gateways Driver Message Rate $\lambda_{jk}^{Gw}$ used in (5) is defined as the total number of messages $n_{jk}$ sent and received by driver $k$ on gateway $j$ in a certain time interval $\Delta t$ and is measured in messages per second ($msg/sec$). The Driver Load $L_k^D$ in (5) is described as the %CPU used by driver $k$ to communicate with the devices connected to the gateway through the driver for a certain message rate and is specific to each device driver.

The power consumption of a server or gateway can be a factor CPU Load, IO Rates, Storage, Memory Accesses and other peripherals. If we consider a case where through migration the resources and drivers are accessed from the same location the main components that we would care about would be the CPU Load and added communication between gateways. If we consider as we did in Reliability that the communication can be partially accounted for by its effect on the CPU we can consider as in [38] that the Power Consumption can be defined as a factor of the Gateway CPU.

*4.2. Application Load*

The total load of an application can be modelled based on our test data from the total CPU usage of the application threads and the known messaging overhead added by the driver and the container broker. These units are defined based on our test application set but would describe any measurable application deployed on the gateway that functions in a similar manner.

$$L_{ij}^A = \frac{(L_{ij}^u + L_M)\lambda_{ij}^A}{P_j^{Cap}} \tag{6}$$

The Total Application Load $L_{ij}^A$ in (6) defines the weight of the application on the Gateway j which is defined as the product of the application unit load $L_{ij}^u$ and the Message Load $L_M$ adjusted to the processing capacity coefficient $P_j^{Cap}$ and multiplied with the application message rate $\lambda_{ij}^A$.

The Message Load $L_M$ in (7) is a constant that denotes the processing impact of the received messages of an application and is the load created by the broker driver between the messaging service and the containers event service. $L_{ij}^a$ and $\lambda_{ij}^A$ in (7) are measured by the monitoring application while $L_{ij}^u$ and $L_{ij}^A$ are calculated.

The Application Message rate or $\lambda_{ij}^A$ used in (6) is defined as the total number of messages $n_{ij}$ sent and received and application i on gateway j in a certain time interval.

$$L_{ij}^u = \frac{L_{ij}^a}{\lambda_{ij}^A}\ P_j^{Cap} \tag{7}$$

The Application Unit Load $L_{ij}^u$ is defined as the reference processing power used to process one message of the application regardless of the current Message Rate. The Unit Load in (7) is used to compare the behavior of different types of applications based on their processor use and message rate, without knowing anything about how they work. We consider if the processing power of the host remains the same, the $L_{ij}^u$ of the application is constant indifferent of deployment location. The unit load is measured in % of the CPU usage for the specified message rate or ($\%cpu/msg/sec$)

The Measured Application Load $L_{ij}^{a}$ in (7) is defined by the amount of CPU the application threads are using on average when running and is denoted by % of the total CPU, as measured by the container monitor.

*4.3. Delay Model*

The delay of the application can be modeled based on the load of the gateway they are on, the number of messages they receive and the amount of load they generate on the gateway. Our model limits its predictability to applications that only perform major processing tasks based on device messages and do not perform background operations. This method can be used for a general application set but its predictability decreases by the amount that the test application differing from our test model.

The Application Delay $D_{ij}^{A}$ can be formulated in (8) as the sum of all application related delays, and is the sum of the processing delay $D_{ij}^{P}$ and the networking delay $D_{ij}^{N}$.

$$D_{ij}^{A} = D_{ij}^{P} + D_{ij}^{N} \tag{8}$$

The Processing Delay $D_{ij}^{P}$ in (9) deducted from out experimental data in the next section, where it is a function of the Gateway Load $L_{j}^{Gw}$ in (4), the Unit Load $L_{ij}^{u}$ defined in (7), the Gateway processing speedup coefficient $P_{j}^{Speed}$ and the two constants $k1$ and $k2$. The values of $k1$ and $k2$ are analogous to a reference $IPS^{Gw}$ value and how it changes with load. The value of $k1$ defines the reference $IPS$ rate of the gateway with no load while $k2$ defines how the $k1$ value is reduced with added load. These reference values are adjusted to differenciate in gateway processing capabilities using $P_{j}^{Speed}$.

$$D_{ij}^{P} = L_{ij}^{u}\ P_{j}^{Speed}\ (k1 + k2\ L_{j}^{Gw}) \tag{9}$$

The Networking Delay $D_{ij}^{N}$ can be described by the sum of delays generated by routing messages from one gateway to another through the messaging system and is described in (10).

$$D_{ij}^{N} = D_{Base}^{R} + \sum_{k=G_{Migrated}}^{j=G_{Host}} D_{Gj,Gk} + D_{Ext}^{R} \tag{10}$$

$D_{Gj,Gk}$ represents the network latency between $j$ and $k$ while the sum is the set of delays that are required to link two gateways. If the application is deployed on the same gateway as the devices, then $D_{ij}^{N} = D_{Base}^{R}$.

The External Routing Delay $D_{Ext}^{R}$ in (10) is a component of the Networking Delay and is an experimentally derived platform specific constant that denotes the average time it takes for a message to be routed from one gateway to another, not taking the latency inside the network into account. The Base Routing Delay $D_{Base}^{R}$ is the delay added to any message going from the driver to either local or external application, and is a fixed experimental and platform specific value.

The Ping Delay $D_{j,k}^{Ping}$ in (10) represents the latency between two directly connected gateways and is measured by the monitoring component on the gateway.

$$D_{Tot} = \sum_{j=1}^{N} \sum_{i=1}^{M} D_{ij}^{A} \tag{11}$$

The Overall Delay of the system $D_{Tot}$ is described in (11) as the sum of all application Delays defined in (8) on the system.

### *4.4. Reliability Model*

For the reliability model we only consider the influence of the Total CPU load on the Reliability of the system, because driver message rates and access values are unchanged when migrating.Furthermore, we consider that the decreased reliability due to added message rates between gateways is covered by the increased Gateway Load $L^{G}w_{j}$.

We adapt the load-hazard model in [39] which provides an analysis of the impact of Load on the probability of the machine to run without any cpu or system errors. They propose a load dependent hazard or failure rate model $z(x)$ , where $x$ is the gateway load $L_j^{Gw}$ . They prove statistically that there is a correlation between increased load and increased failure rates. They define the hazard rate $z(x)$ as the probability of an error occurring at a cpu load $x + \Delta(x)$,where $\Delta(x)$ is the added load ,given that there was no failure at load $x$.

Considering (12) for defining reliability, where $R(t)$ is a function of the hazard function $z(t)$ or a function of the constant failure rate $\lambda$. In the case of the hazard function we can replace $\lambda$ with $z(L_j^{Gw})$ as we consider it constant in time as well.

$$R(t) = e^{-\int_0^t z(u)du} = e^{-\lambda t} = e^{-z(L_{Gw})t} \tag{12}$$

Based on the data from [39] the hazard function $z(L_j^{Gw})$ is approximated to a third-degree polynomial (13) where $x$ is between 0.08 and 0.96 in CPU usage, the min value is 0.0018 and the max value 0.0118

$$z(x) = 0.0195x^3 - 0.137x^2 + 0.0059x + 0.0015 \tag{13}$$

Considering a constant runtime of a day or 24 hours we can define our Gateway Reliability $R_j^{Gw}$ using the Load based Reliability function $R(L_j^{Gw})$ in (14). This gives us a maximum reliability of 95.5% and a minimum reliability of 75.35%.

$$R_j^{Gw} = e^{-0.468{L_j^{Gw}}^3 + 0.3288{L_j^{Gw}}^2 - 0.1416L_j^{Gw} - 0.036} \tag{14}$$

## 5. Optimization Methods

For our optimization, we consider the scenarios of an agile industrial environment with certain Delay and Reliability requirement. Based on this we consider the main parameters for our optimization to be the delay between gateway messages and the time it takes to process them and the load of the gateway which affects reliability. We consider all other messages to cloud, region, resources and peer applications as non-vital messages.

Load Balancing in the context of Fog Computing is mostly focused on optimizing the load distribution between peers. In the context of CPS, we are particularly interested in individual messages generating a response within an allocated time as a real-time constraint, rather than having processed 100 messages in a certain time. Furthermore, different applications may have different QoS requirements and constraints regarding latency.

### *5.1. Optimization Problem Description*

Our methods are different from traditional load balancing and delay optimizations methods, as the proposed DO method focuses on reducing the single message delay time for all applications on the system and the ConstDO method suggests reducing this delay but attempting to satisfy the latency requirements of key applications first.

We aim to show that methods that improve certain variables might not be suitable for others. The Reliability of the gateways increases exponentially with the Gateway Load while the Power consumption increases linearly with the Gateway Load.

Using the application delay $D_{ij}^{A}$ and the Gateway Load $L_{j}^{Gw}$ we define four fitness functions $Cost(G, A)$ for the application set $A$ on Gateways $G$ to be optimized.

The first approach looks at increasing the reliability of the system, which due to the exponential nature of the load-reliability relation can be achieved by reducing the load variation between gateways. We define the fitness function for the Reliability Optimization (RO) method (15) as the product of Gateway Reliability in the cluster due to the series type dependence between them.

$$Cost(G, A) = \prod_{j=1}^{n} R_{j}^{Gw} \tag{15}$$

The DO method based on (16) looks at reducing the overall delay of the cluster, which we define as the sum of all application delays $D_{ij}^{A}$ in (11) from all the Gateways:

$$Cost(G, A) = \sum_{i=1}^{n} \sum_{j=1}^{m} D_{ij}^{A} \tag{16}$$

The ConstDO method based on (17) looks at reducing the overall delay of the cluster while maintaining or reducing the number of applications that exceed their constraints. This can be done in two ways. We define $Ct_{ij}^{A}$ the specific Delay constraint for application i on gateway j. The first method used is the Hungarian method which allows for multiple cost functions while the second one is the Genetic Algorithm (GA) which needs a fitness function that returns only one value. The function in (18) modifies (17) to accommodate this.

$$Cost(G, A) = \left[ \sum_{i=1}^{n} \sum_{j=1}^{m} D_{ij}^{A}; \sum_{i=1}^{n} \sum_{j=1}^{m} D_{ij}^{A} > Ct_{ij}^{A} \right] \tag{17}$$

We achieve this by giving a certain delay to the applications that violate their constraints. This is done by modifying the fitness function by adding a delay equivalent to the number of applications over the constraint multiplied by a $W_{Ct} = 100$.

$$Cost(G, A) = \sum_{i=1}^{n}\sum_{j=1}^{m} D_{ij}^{A} + W_{Ct}\sum_{i=1}^{n}\sum_{j=1}^{m} D_{ij} > Ct_{ij}^{A} \tag{18}$$

*5.2. Optimization Methods*

We use the different optimization algorithms with varying initial scenarios to assure that the results of the optimizations reflect the characteristics of the fitness functions and not the algorithms. The ConstDO and DO approaches have the best results when it comes to the specific delays of applications, while the RO approach may cause added networking delay due to unnecessary or counterproductive migration.

**ALGORITHM 1:** Modified Hungarian Algorithms

```
1  Set iter ← 0; minCost ← Inf; iniCost ← Inf;
2  while minCost ≤ iniCost and maxInter ≤ inter do
3  |  IniCost = Cost(G,A); iter ← iter + 1;
4  |  for g in G do
5  |  |  forall a in A where a ∉ g do
6  |  |  |  MoveApp a → g;
7  |  |  |  projCosts ← Cost(G,A);
8  |  |  |  if projCosts ≤ minCosts then
9  |  |  |  |  minCosts ← projCosts;
10 |  |  |  |  min[g, a] ← [g, a];
11 |  |  |  undo MoveApp a → g;
12 |  iter ← iter + 1;
13 |  if  minCosts < iniCost then
14 |  |  MoveApp min[a] → min[g];
```

Optimizing the deployment of multiple applications to multiple gateways, discussed in this paper, is an assignment problem, similar to multiple knapsack problems. Such generalized assignment problem (GAP) is an NP-hard problem. Suppose that, we take a greedy approach as in [40] and search the complete space of solutions. In such case, having n gateways and k applications we would need to analyse alternative solutions. An extended review of GAP and its applications was presented in [41] . Methods proposed to solve GAP can be classified as exact and approximation (heuristic) algorithms. Both types of algorithms for solving the assignment problem assume a priori existence of a matrix of edge weights,$w_i$, or costs, $c_{ij}$, and the problem is solved with respect to these values.

The first method applied to the problem discussed in this paper, is a modified Hungarian method [42] extended to many-to-many assignment problems shown in Algorithm 1. It takes initial allocation of m-applications to n-gateways, calculates its $Cost(G, A)$ function based on (15),(16) and (17). The next step is to find the lowest value and compare it with the load cost of the current assignment. If the current assignment load cost is higher or equal to the lowest value the configuration parameters are saved. These steps are repeated until a more optimal assignment is no longer possible or the maximum iteration is reached.

**ALGORITHM 2:** Modified Genetic Algorithm

```
Set popSize ← 500; genMax ← 1000; pop ← genPop(popSize);
while genCount ≤ genMax do
    sortPop ← sort(pop , Cost(G,A));
    newPop ← sortPop[1 ... popSize*0.2];
    newPop ← newPop + genPop(popSize*0.3);
    for [indi1,indi2] in sortPop[1 ... popSize*0.6] do
        for i = 0 to size(indi1) do
            newIndi[i] ← randomSelect(indi1[i],indi2[i]);
        newPop ← newPop + newIndi;
    for indi in pop do
        if random() ≤ 0.2 then
            indi[random(1,len(indi))]← rand(1,gwCount);
            newPop ← newPop + indi;
    pop ← newPop
```

Second method used in Algorithm 2 is a modified Genetic Algoirthm that uses a one-dimensional chromosome, where the i-th position in the chromosome represents the i-th application and the encoded gene represents its allocated gateway. The optimization starts with an initial population of 500 randomly generated strings and iterates over a 1000 generations while implementing natural selection of best fit strings. The tournament selection is used to preserve the good solutions. The crossover rate is 50% chance for two selected parents to swap parts of their genes and the mutation rate is based on the inverse of the total number of applications. The $Cost(G, A)$ function that were used are described in (15),(16) and (18).

## 6. Parameter Analysis

We perform our tests in a physical environment based on the scenario in [22] presented in Section 1 to find the parameters of our application and gateway model . Our testing environment consists of homogeneous Raspberry Pi nodes that have a processing speedup coefficient of $P_j^{Speed}$ and processing capacity coefficient $P_j^{Cap}$ of 1. When migrating to the cloud we have two medium flavored VM's on different hosts with varying processing capabilities.

### *6.1. Processing Capacity and Speedup*

The Gateway processing speedup $P_j^{Speed}$ and processing capacity $P_j^{Cap}$ are calculated in (2) and (3) by deploying a reference application for 5 minutes and measuring a stable 4 minutes the load it creates and delays of the application. For our testing we deploy a set of 5 applications with varying loads on top of 4 types of gateways, two VM's and two Raspberry Pi's. The first VM (VM1) has 2 VCPU's and 4GB of RAM on a host with an i5-Intel Xeon E312 3.1Ghz Processor with a Bogomips value of 6185.94 . The second VM (VM2) has 2 VCPU's and 4GB of RAM on a host with an i7-4770 3.4GHz Processor with a Bogomips value of 6784.28. The raspberry pi used is a nr. 2 model b with an ARM

Cortex-A53 1.2 GHz quad core processor and 1GB of ram . Two scenarios are considered with the Raspberry pi, for the first (RPi1) its overclocked to 1.2GHz and for the second (RPi2) its left at the base value of 1GHz. This overclocking changes the Bogomips value of the pi from 697.95 to 732.2. The more performant VM and Raspbery pi is used for the rest of the testing and evaluation.

For the estimation of the processing capacity $P_j^{Cap}$ values we measure the systems CPU use when on idle giving us an $L_{Idle}$ of 3.68% for the Raspbery Pi's and 1.8% for the VM's. After this we deploy our load application with varying load. This application mimics our standard applications but it performs the processing tasks on a timer rather then on received messages making the created load more stable. The deployed loads were 0, 50, 100, 200, 300, 500, and 1000 cycles of the processing task. We measure the CPU load of the Machines and based on (3) we get the processing capacity of the machines.

For the estimation of the processing speedup $P_j^{Speed}$ we deploy an application on the machines and measure its Processing Delay $D_{ij}^P$. We account for the differences in processing capacity of the systems by deploying the load app to the more performant systems to match the load of the others. The application deployed was given 50,100,250,500 and 700 cycles of the processing task with a message rate of 5 giving us unit load $L_{ij}^u$ values of 0.55, 1.008, 2.448, 4.76 and 6.6126. We measure the processing times of these applications and based on (2) we get the processing speedup of the machines.

Table 2: Processing parameters of the Machines

| Machine | VM1 | VM1 | RPi 1 | RPi2 |
|---|---|---|---|---|
| Processor | i7-4770 3.4 GHz | I5-Intex Xeon 3.1 Ghz | ARM Cortex-A53 1.2 GHz | ARM Cortex-A53 1.0 GHz |
| CPU Count | 2 | 2 | 4 | 4 |
| BogoMips | 6784.28 | 6185.94 | 732.2 | 697.95 |
| $P_j^{Cap}$ | 2.556 | 2.304 | 1 | 0.924 |
| $P_j^{Speed}$ | 4.307 | 4.189 | 1 | 0.874 |

From these experiments we get the results in Table 2., where we can see the differences in the machines and that while the BogoMips is a good indicator of the direction of the processing coefficients it can't be used to estimate these.

*6.2. Driver and Message Loads*

We measure the loads of the drivers $L_k^D$ on our system by measuring the idle load of the system with the drivers running but no messages being sent through, after which we send messages at different rates to a non-routable queue and measure the added load on the CPU. We divide the results with the message rates and get the driver characteristic load for

each message received. We test this for the RF24 , Bluetooth and the TestingDrivers. The tested message rates were 1, 3, 5, 10 and 20 messages every second.

To measure the load of routing the messages on the system and through the Karaf container we use our TestingDriver to send messages to an application with a know Unit load $L_i^u j$ of 1.008 with varying message rates of 1,3,5 and 10. We measure the application load and calculate the base load $L_j^B$ based on the message rates, initial idle values and the based on (6) we calculate the value of $L_M$. We take the mean value of the 4 tests and that gives us the reference value.

For the Driver Loads $L_k^D$ we get averages of 1.61 for each RF24 message, 0.73 for each Bluetooth Message and 1.10 for the Testing Driver. The difference in created load can be attributed to the implementation of the drivers but also the hardware support on the system. While the bluetooth driver has a dedicated chip that takes off some of the load by implementing more of the OSI model on chip while for the RF24 most of this needs to be done by the driver. For our driver, the load is caused by the measurements and data retention. The average messaging load $L_M$ on the system was found to be 0.285.

### *6.3. Processing Delays*

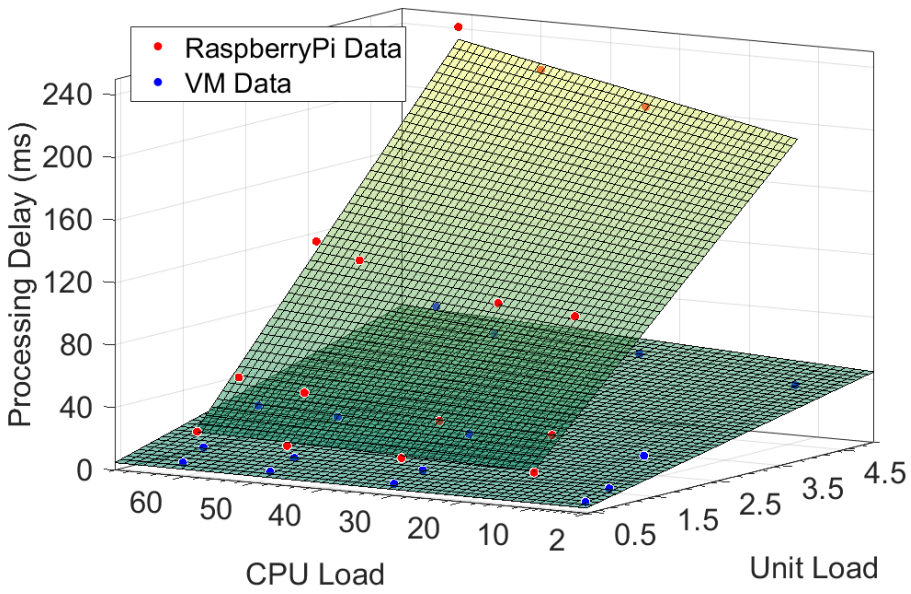



Figure 3: Processing Delay Variation

To estimate the processing delays $D_{ij}^P$ of the system we need to find k1 and k2 which represent the characteristic delay for a certain amount of processing done with a certain amount of load on the system. From our tests we have concluded that the RAM of the system only comes into play when we go above 80% ram use, in which case the system may crash. Furthermore, the message rate only influences every individual message by adding load to the system, and when that contribution is taken out no further delay is added. If

the message rate exceeds the Karaf driver capacity or the application capacity we consider an overload situation. In conclusion, for our system we test the impact of the Gateway Load $L_j^{Gw}$ and the Application Unit Load $L_{ij}^u$ within normal operation parameters of the system where RAM use doesn't exceed 80% and the messaging rate doesn't cause bottlenecks. We can test for these through the Gateway monitoring application for RAM and the RabbitMQ monitoring for queued up messages to validating the message rate.

For the testing we deployed 4 known application with the unit loads of 0.55,1.008,1.906 and 4.76 and a constant messaging rate of 5 messages every second. We measured the total CPU of the system and the Application load through the monitoring component. To account for different scenarios we increased the CPU usage of the gateways through the Load app. We deployed this scenario on VM1 and RPi1 and got the points presented in fig. 3. Using the equation from (9) and the performance values from table 2 we do a curve fitting to match the data-points, having k1 and k2 as unknowns. The resulting function gives us the slightly curved form of the two surfaces.

$$\begin{aligned} D_{ij}^P &= L_{ij}^u \; P_j^{Speed} \; (38.409 + 0.1885 \; L_j^{Gw}) \\ D_{i\;VM1}^P &= L_{ij}^u \; (9.169 + 0.0459 \; L_{VM1}^{Gw}) \\ D_{i\;RPi1}^P &= L_{ij}^u \; (38.409 + 0.1885 \; L_{RPi1}^{Gw}) \end{aligned} \tag{19}$$

The resulting equations for $D_{ij}^P$ in the general case, for VM1 and for RPi1 can be seen in (19). The value for k1 is 38.409 and 0.1885 for k2 where the first means that for each unit of load we add 38.409 milliseconds of processing on a reference system while k2 means that for each percentage of extra load we add 0.1885 milliseconds of processing delay for each unit of load on the reference system. If we consider the processing speedup $P_j^{Speed}$ of the machines the PRi1 stays the same while for the Vm this changes to 9.169 and 0.0459.

### *6.4. Networking Delays*

We measure the Networking Delay by testing the response time of an application when running on the RPi1 and when running on a VM1. Our monitoring component measures the ping values between peers so we compare the response times to the ping values. The latency between the two gateways was increased using netem [43] for Linux from 0 to 80ms to match typical cloud-user latencies.

$$D_{ij}^N = 9.83 + \sum_{k=G_{Migrated}}^{j=G_{Host}} D_{Gj,Gk} + 8.246 \tag{20}$$

To get the value of the base Routing Delay $D_{Base}^R$ and that of the External Routing Delay $D_{Ext}^R$ we subtract the processing delay from the total delay which gives us to total $D_{ij}^N$. From this we subtract the measured ping value $D_{Gj,Gk}$ between the two gateways which gives us $D_{Base}^R + D_{Ext}^R$. We then subtract the mean values for local processing where only $D_{base}^R$ is present with a value of 9.83ms. The resulting $D_{Ext}^R$ has a mean of 8.246ms giving us the equation for $D_{ij}^N$ in (20).

The graphs in fig. 4 shows the impact the networking delay has on the total delay, giving us a set of gateway load $L_j^{Gw}$ and Unit Load $L_{ij}^u$ threshold value where applications have smaller latencies on the gateway than on the cloud VM. If we consider the P1 points on the figure we can see the diffrence in their values between processign and total delay on both the VM and the RPi. For these cases the points are a result of having an app with a $L_{ij}^U$ of 1.906 and a gateway load $L^G w_j$ for the RPi of 51.917% and 51.425% for the VM. This gave us a calculated Total Delay of 101.69ms for the RPi and 90.57ms for the VM. The actual values were 108.743ms for the RPi and 85.051ms for the VM. This gives us a mean error of 6.29%.

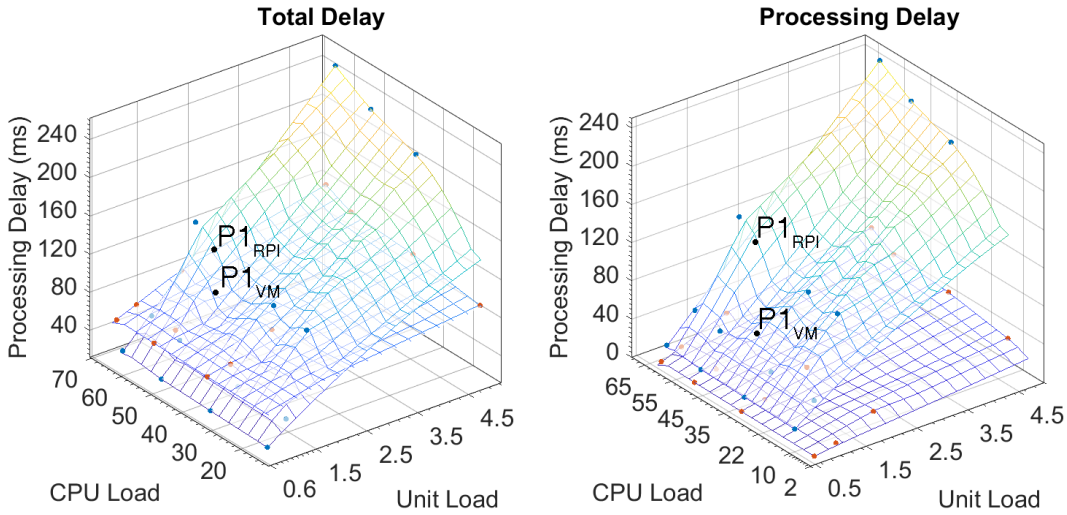



Figure 4: Application Delays variation

## 7. Model Validation and Testing

For these tests we propose a set of applications that have different loads and message rates. We tested them individually on the gateways, deployed them as groups and migrated components to verify that the presented application model stands and that the estimations for Gateway Load and Application Delay are correct. Based on these we measure the accuracy of our model.After the model is verified and the resulting parameters tested, we perform two sets of optimization tests.

The first set consists of testing the optimization of a system of four gateways having a varying set of applications deployed on them. We measure the run-time parameters, validating them to the model, then run the optimization algorithms, deploying the results and validating them on the physical system. Based on these tests we can say how well the two approaches work on a small-scale deployment.

In the second set we test the scalability of the methods and optimization algorithms by generating a random set of gateways with a set of available cloud VMs with random latencies

between them and a random set of deployed applications and look at how well each method performs.

The main assessment criteria for our model is the precision of the Load and Delay estimation on the system. For the optimization methods, we assess them by looking at the decrease in application delays and the reduction of constraint violations that are adjusted to the size of the cluster by looking at how much the total delay drops on average for each gateway as well as how many applications can fulfill their requirements on average per gateway.

### *7.1. Single and Bundled Model validation*

The initial models error is known from the fitting tests which gave us 7.34%. A more extensive test is performed and the results evaluated when deploying to four RPi1 type gateway and an equivalent cloud GW. We have designed 8 applications with varying loads if 0.465, 1.45, 4.265 and 9.915 and varying message rates of 1, 4 and 20. This test consists of deploying applications on a free gateway, measuring the app and the gateways run-time parameters and comparing them to the estimations that the application model made.The Idle Load $L_j^{Idle}$ in (5) for these tests is 5.64. Our estimations of a single app deployment had an average error for the Total Load of 3.30% and 4.99% for the Estimated Delay, with a maximum error for the load of 5.68% and 8.35% for the delay. These maximum values were achieved at the edges of the linearization by two apps. For further testing we consider a remainder of 9 apps that lacked the maximums.

Bundled deployment testing considers the applications from the first test and deploys 10 sets of them on the gateway and verifies how accurate the model is in determining the application delay and the total processor use. For these tests, we consider the applications having the same Unit Load as determined previously.

From these tests, we concluded that the mean Gateway Load estimation error when deploying a set of applications is 3.92% while the maximum value is 8.6% which was found for the deployments with applications of low message rates. The Delay estimation error was found to have a mean of 5.47% and a worst result of 9.04% from the same set of applications.

For the final tests, we migrate an application, measure the changes in the original host of the application and consider the delay time of the application and how accurate this estimation was. The resulting errors measure the estimation error correlated to the total value of the delay. In these cases, we got an average estimation error of the CPU of 2.26% with a maximum of 4.91. The delay estimations had an average error in accuracy of 1.75% with a peak of 4.4%. This is partially due to the small size of the errors compared to the total delay as well as using monitored parameters to estimate the results of the migration.

### *7.2. Optimization on Physical Devices*

We consider our Fog of Things Platform in [36] having 4 gateways and a Virtual Cloud Gateway as our system. The gateways have a delay between them of an average of 19.53ms while the delay between the cloud gateway is set to an average of 42ms. The initial state of the cloud is empty only having application migrated to it after optimization if needed. We generate a set of initial configurations which we deploy and monitor on the gateways. After

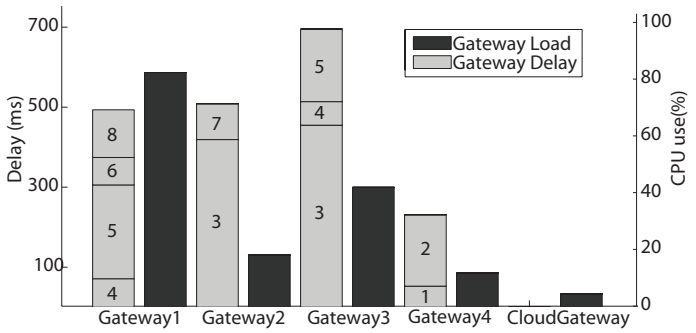
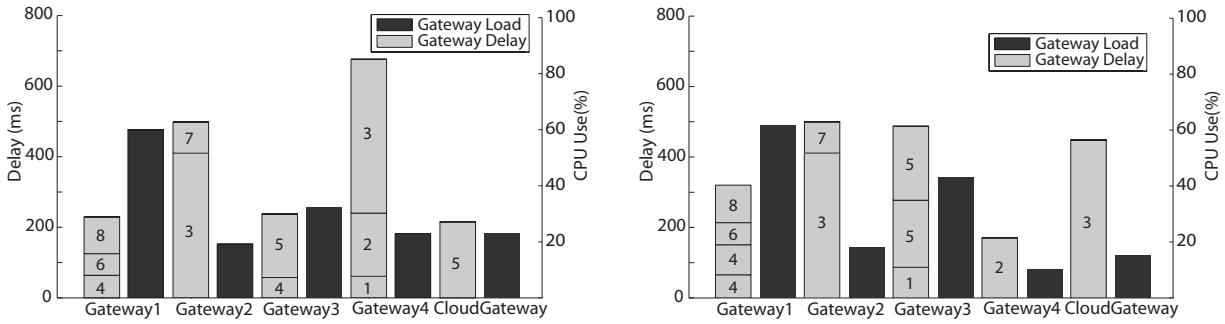



Figure 5: Initial Deployment

that we feed the information into the optimization algorithms that come up with the best solution within the parameters and proposed functions.




(a) Delay Optimization (DO)

(b) Constraint Delay Optimization (ConstDO)

Figure 6: Delay Optimization Results

We then take the results and test them on the physical cluster and examine what the actual values of the proposed new configuration are. The initial Delays and CPU usage of the gateway can be seen in Fig 5. The total delay of the system is 1881.80ms while the average Load variation is 21.67% with a System Reliability of 66.53% and there are 2 applications that do not meet their constraints.

For this initial phase, we are interested in the differences in results from the given fitness functions. For this set we used the best results for each function from the Hungarian, GA and Random methods. The results can be seen in fig. 7 for RO and fig. 6 for DO and ConstDO.Here the number represents the application ID.

From these tests, we can see that while the Reliability Optimization method managed to reduce the load variation to 2.07% and gave us the maximum reliability of 73.56%. This didn't improve constraint violations and it actually increased the delays to 2038ms due to unnecessary migration. The DO achieved a minimum total delay of 1854ms with 1 constraint violation while the ConstDO had a higher total delay of 1974.1 ms but managed to have

0 constraint violations. Both the DO and ConstDO methods improved the Reliability to 72.35% and 71.43% which is an improvement to the initial deployment but falls short of the RO results.

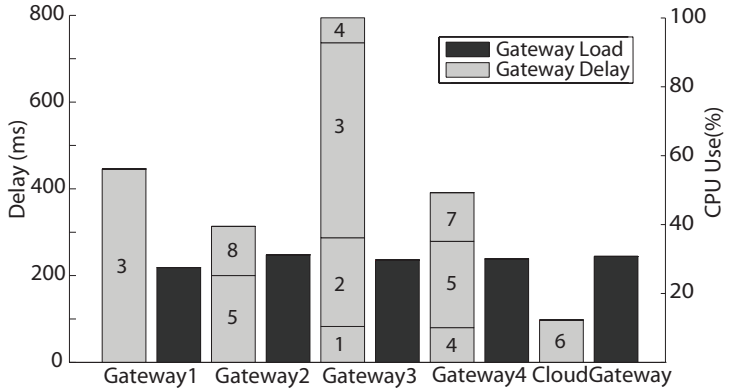
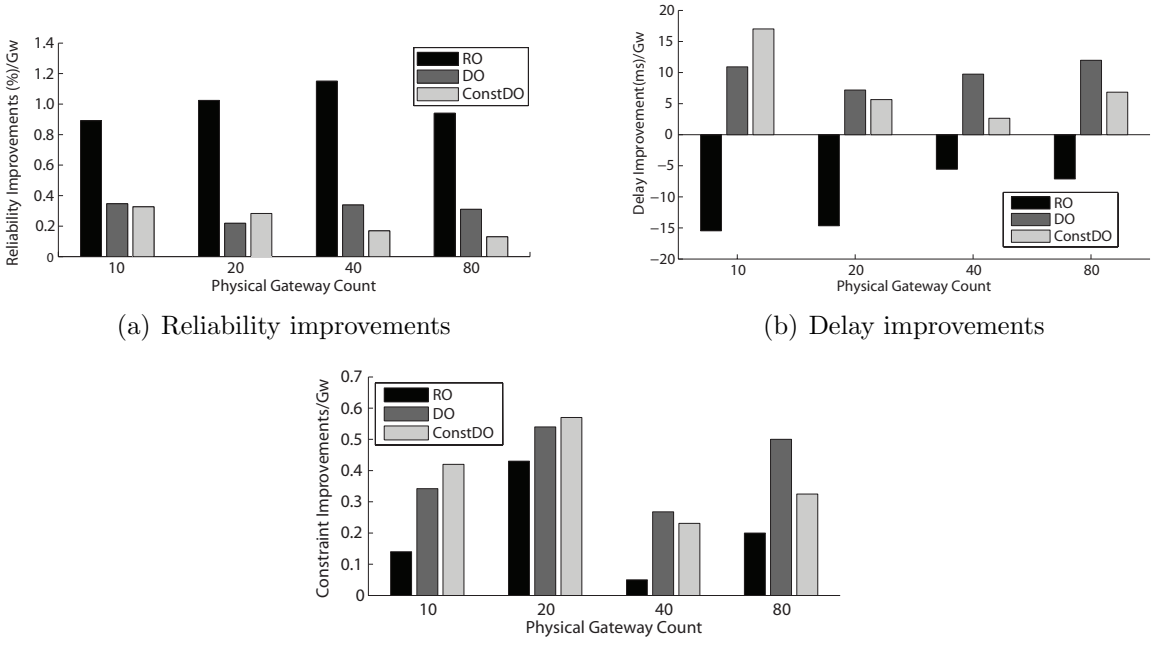



Figure 7: Reliability Optimization Results

### *7.3. Scaling Simulation of the Optimization Algorithms*




(a) Reliability improvements

(b) Delay improvements

(c) Constraint Violation Improvements

Figure 8: Scaling Tests

The scaling tests are designed to show how these methods behave when scaling the cluster size. Such scaling would result in the increase of the base load of a system. For our testing, we generate a set of gateways with the existing parameters.

We generate 5 deployments of 10, 20, 40 and 80 gateways and look at the resulting reduction in the total Delay and the number of applications over their constraints. The results of these tests can be seen in Fig. 8(a) for the Reliability, Fig. 8(b) for the constraints violations and Fig. 8(c) for the Delay. We consider the average results of 5 test for all approaches throughout the scaling. To adjust for the size of the clusters we divide the resulting improvements over the total gateway count.

We can see from the results of the experiments that the behaviors from the physical tests are maintained throughout the scalability tests where the best results based on delay are reached by the DO algorithms while the ConstDO is best at reducing the constraint violations and the RO is best at increasing the reliability of the system.

These tests also show that with the increased scale of the system, the capability of all methods to find good solutions decreases. With this increase in size the processing times increase exponentially as well. Global optimization methods designed for large-scale Fog environments need to account for both the interconnectivity and heterogeneity of these systems, but should also consider their scale.

## 8. Conclusions and Future Work

When considering CPS and Industrial environments, the latencies of a system and its reliability are crucial components. Estimation and Optimization attempts in Fog Computing and IoT need to consider as a complete model as possible for their methods so they do not lose applicability and accuracy.

Our model and platform provides a way of measuring and estimating the runtime parameters and migration benefits of applications in such systems. Based on our model we propose a novel approach to load and delay optimization through application migration between the Edge and the Cloud. These allow us to estimate and improve certain parameters of deployed applications. Inspired by [44], an experimental load model description derived from measuring runtime parameters over physical systems has been developed and used to represent the gateway and application loads, which provide a more realistic estimation than theoretical ones presented in other papers.

The experimental results have shown that the system has an overall accuracy of over 90% for both the Delay and Load models. Furthermore, testing on physical systems has shown that even in different scales for both the physical and virtual environment, our proposed methods provided improvements for applications meeting their constraints and reduced the overall delay of the system compared to the initial deployment scenario. The results also show that it has outperformed the Load variation based Reliability Optimization Method in terms of delay improvements, but pared on the reliability. Most importantly the Constraint based optimization method managed to clear all constraint violations for the physical system and reduced these by an average of 67% for the scaling tests.

The assignment problem that results from attempting to minimize the proposed methods is an NP hard placement problem with interdependent parameters that has proven to be a challenging one for both heuristic and deterministic methods, neither being able to provide the best results in all cases. A drop in the improvements of the provided by the methods

can be seen with the increase of scale, this is cause by the exponential increase in the search domain of possible solutions. This suggests that proposals are required that can account for the challenges resulting from the scale and heterogeneity of fog systems In future work we will examine how these can be improved and how the complexity of the problem can be reduced.

In our future work we will expand our methods to include migration costs, and other parameters. We will look at combining the delay and reliability methods to allow gateway and app profile based optimization. Finally, we will be looking at advanced methods of optimization together with varying techniques and objective functions to improve various aspects of deployment.

## References


[1] J. Lee, B. Bagheri, H.-A. Kao, A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems, Manufacturing Letters 3 (2015) 18–23. doi:http://dx.doi.org/10.1016/j.mfglet.2014.12.001.

[2] DIN, German Standardization Roadmap Industry 4.0 (Version 2) (2016).

[3] Y. Lu, Industry 4.0: A survey on technologies, applications and open research issues, Journal of Industrial Information Integration 6 (2017) 1–10. doi:10.1016/j.jii.2017.04.005.

[4] A. J. C. Trappey, C. V. Trappey, U. H. Govindarajan, J. J. Sun, A. C. Chuang, A Review of Technology Standards and Patent Portfolios for Enabling Cyber-Physical Systems in Advanced Manufacturing, IEEE Access 4 (2016) 7356–7382. doi:10.1109/ACCESS.2016.2619360.

[5] L. Wang, M. Törngren, M. Onori, Current status and advancement of cyber-physical systems in manufacturing, Journal of Manufacturing Systems 37, Part 2 (2015) 517–527. doi:http://dx.doi.org/10.1016/j.jmsy.2015.04.008.

[6] S. Wiesner, E. Marilungo, K.-D. Thoben, Cyber-Physical Product-Service Systems: Challenges for Requirements Engineering (Mini Special Issue on Smart Manufacturing), International journal of automation technology 11 (1) (2017) 17–28.

[7] P. Mell, T. Grance, The NIST Definition of Cloud Computing, Tech. rep. (2011).

[8] R. Chaâri, F. Ellouze, A. Koubâa, B. Qureshi, N. Pereira, H. Youssef, E. Tovar, Cyber-physical systems clouds: A survey, Computer Networks 108 (2016) 260–278. doi:http://dx.doi.org/10.1016/j.comnet.2016.08.017.

[9] Health and Safety Executive, Health and safety in engineering workshops (2004).

[10] R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption, IEEE Internet of Things Journal 3 (6) (2016) 1171–1181. doi:10.1109/JIOT.2016.2565516.

[11] Y. Jiang, A Survey of Task Allocation and Load Balancing in Distributed Systems, IEEE Transactions on Parallel and Distributed Systems 27 (2) (2016) 585–599. doi:10.1109/TPDS.2015.2407900.

[12] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things (2012). doi:10.1145/2342509.2342513.

[13] C. T. Do, N. H. Tran, Chuan Pham, M. G. R. Alam, Jae Hyeok Son, C. S. Hong, A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing, in: 2015 International Conference on Information Networking (ICOIN), IEEE, 2015, pp. 324–329. doi:10.1109/ICOIN.2015.7057905.

[14] R. Mahmud, R. Buyya, Fog Computing: A Taxonomy, Survey and Future DirectionsarXiv:1611.05539.

[15] M. Díaz, C. Martín, B. Rubio, State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing, Journal of Network and Computer Applications 67 (2016) 99–117. doi:10.1016/j.jnca.2016.01.010.

[16] I. Stojmenovic, Fog computing: A cloud to the ground support for smart things and machine-to-machine

networks, in: 2014 Australasian Telecommunication Networks and Applications Conference (ATNAC), 2014, pp. 117–122. doi:10.1109/ATNAC.2014.7020884.

[17] L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, Cost Efficient Resource Management in Fog Computing Supported Medical Cyber-Physical System, IEEE Transactions on Emerging Topics in Computing 5 (1) (2017) 108–119. doi:10.1109/TETC.2015.2508382.

[18] F. Tao, Y. Cheng, L. D. Xu, L. Zhang, B. H. Li, CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System, IEEE Transactions on Industrial Informatics 10 (2) (2014) 1435–1442. doi:10.1109/TII.2014.2306383.

[19] S. K. Khaitan, J. D. McCalley, Design Techniques and Applications of Cyberphysical Systems: A Survey, IEEE Systems Journal 9 (2) (2015) 350–365. doi:10.1109/JSYST.2014.2322503.

[20] A. James, J. Cooper, K. Jeffery, G. Saake, Research directions in database architectures for the internet of things: a communication of the first international workshop on database architectures for the internet of things (dait 2009), Dataspace: The Final Frontier (2009) 225–233.

[21] Y. Nikoloudakis, S. Panagiotakis, E. Markakis, E. Pallis, G. Mastorakis, C. X. Mavromoustakis, C. Dobre, A Fog-Based Emergency System for Smart Enhanced Living Environments, IEEE Cloud Computing 3 (6) (2016) 54–62. doi:10.1109/MCC.2016.118.

[22] N. Verba, K.-M. Chao, A. James, J. Lewandowski, X. Fei, C.-F. Tsai, Graph analysis of fog computing systems for industry 4.0, in: 2017 IEEE 14th International Conference on e-Business Engineering (ICEBE), IEEE, 2017, pp. 46–53.

[23] M. García-Valls, T. Cucinotta, C. Lu, Challenges in real-time virtualization and predictable cloud computing, Journal of Systems Architecture 60 (9) (2014) 726–740. doi:http://dx.doi.org/10.1016/j.sysarc.2014.07.004.

[24] L. D. Dhinesh Babu, P. Venkata Krishna, Honey bee behavior inspired load balancing of tasks in cloud computing environments, Applied Soft Computing 13 (5) (2013) 2292–2303. doi:http://dx.doi.org/10.1016/j.asoc.2013.01.025.

[25] G. Lee, B.-G. Chun, H. Katz, Heterogeneity-aware resource allocation and scheduling in the cloud (2011).

[26] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente, Scheduling strategies for optimal service deployment across multiple clouds, Future Generation Computer Systems 29 (6) (2013) 1431–1441. doi:http://dx.doi.org/10.1016/j.future.2012.01.007.

[27] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, Y. Li, Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches, ACM Comput. Surv. 47 (4) (2015) 1–33. doi:10.1145/2788397.

[28] J. Hu, J. Gu, G. Sun, T. Zhao, A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment, in: 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming, 2010, pp. 89–96. doi:10.1109/PAAP.2010.65.

[29] J. Zhao, Y. Ding, G. Xu, L. Hu, Y. Dong, X. Fu, A Location Selection Policy of Live Virtual Machine Migration for Power Saving and Load Balancing, The Scientific World Journal 2013 (2013) 16. doi:10.1155/2013/492615.

[30] X. Wu, M. Deng, R. Zhang, B. Zeng, S. Zhou, A Task Scheduling Algorithm based on QoS-Driven in Cloud Computing, Procedia Computer Science 17 (2013) 1162–1169. doi:http://dx.doi.org/10.1016/j.procs.2013.05.148.

[31] F. Ramezani, J. Lu, F. K. Hussain, Task-Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization, International Journal of Parallel Programming 42 (5) (2014) 739–754. doi:10.1007/s10766-013-0275-4.
URL http://dx.doi.org/10.1007/s10766-013-0275-4

[32] S. Ningning, G. Chao, A. Xingshuo, Z. Qiang, Fog computing dynamic load balancing mechanism based on graph repartitioning, China Communications 13 (3) (2016) 156–164. doi:10.1109/CC.2016.7445510.

[33] S. Verma, A. K. Yadav, D. Motwani, R. S. Raw, H. K. Singh, An efficient data replication and load balancing technique for fog computing environment, in: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 2888–2895.

[34] L. F. Bittencourt, M. M. Lopes, I. Petri, O. F. Rana, Towards Virtual Machine Migration in Fog Computing, in: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015, pp. 1–8. doi:10.1109/3PGCIC.2015.85.

[35] X. He, Z. Ren, C. Shi, J. Fang, A novel load balancing strategy of software-defined cloud/fog networking in the Internet of Vehicles, China Communications 13 (Supplement2) (2016) 140–149. doi:10.1109/CC.2016.7833468.

[36] N. Verba, K.-M. Chao, A. James, D. Goldsmith, X. Fei, S.-D. Stan, Platform as a service gateway for the Fog of Things, Advanced Engineering Informaticsdoi:http://dx.doi.org/10.1016/j.aei.2016.11.003.

[37] M. D. Hill, M. R. Marty, Amdahl's law in the multicore era, Computer 41 (7).

[38] M. Blackburn, G. Grid, Five ways to reduce data center server power consumption, The Green Grid.

[39] R. K. Iyer, D. J. Rossetti, A Measurement-Based Model for Workload Dependence of CPU Errors, IEEE Transactions on Computers C-35 (6) (1986) 511–519. doi:10.1109/TC.1986.5009428.
URL http://ieeexplore.ieee.org/document/5009428/

[40] L. Özbakir, A. Baykasolu, P. Tapkan, Bees algorithm for generalized assignment problem, Applied Mathematics and Computation 215 (11) (2010) 3782–3795. doi:http://dx.doi.org/10.1016/j.amc.2009.11.018.

[41] D. W. Pentico, Assignment problems: A golden anniversary survey, European Journal of Operational Research 176 (2) (2007) 774–793. doi:http://dx.doi.org/10.1016/j.ejor.2005.09.014.

[42] H. W. Kuhn, The Hungarian Method for the assignment problem, Naval Research Logistics Quarterly 2 (1955) 83–97.

[43] S. Hemminger, Network Emulation with NetEm.
URL https://www.rationali.st/blog/files/20151126-jittertrap/netem-shemminger.pdf

[44] T. Kunz, The influence of different workload descriptions on a heuristic load balancing scheme, IEEE Transactions on Software Engineering 17 (7) (1991) 725–730. doi:10.1109/32.83908.